6.26

节省了运行时间，降低时间复杂度

先用 `SPLIT` 算法划分之后，进行排序之后，可以直接取出子序列的中项，找到整个序列的中项，不需要再使用递归算法去寻找中项

6.33

可以改善

使用 `三数取中` 即随机取三个关键字进行排序，然后将中间数所谓枢轴，一般我们选区左端，右端和中间三个数。这样至少中间的数不是最大和最小数。

结果：`a[low]` 左侧都比 `a[low]` 小，`a[high]` 右侧都比 `a[high]` 大

三值取中可以尽量确保数组划分的尽量平衡
在最平衡的划分当中，`PARTITION` 得到的两个子数组的规模都不大于 n/2，一个数组规模为[n/2]，另一个数组的 规模为[n/2]-1，那么快排的性能就会特别好，算计你发运行时间的递归式是 `T(n) = 2T(n/2) + O(n/2)`，此时解为O($nlog_2n$),通过在每一层上都进行平衡划分，我们得到了一个渐进时间上更快的算法。

三值取中算法具体如下：

```
int Split(int a[],int low,int high) {
    int i=low,temp;
    if(a[low]< a[(low+high)/2]) {
        if(a[low]<a[high]) {
            if(a[(low+high)/2]<a[high]) {
                //其值居中者为a[(low+high)/2]
                temp= a[(low+high)/2];
                a[(low+high)/2]=a[low];
                a[low]=temp;
            }
            else {
                //其值居中者为a[high]
                temp=a[high];
                a[high]=a[low];
                a[low]=temp;
            }
        }
    }
    else {
        if(a[(low+high)/2]<a[high]) {
            if(a[low]>a[high]) {
                //其值居中者为a[high]
                temp=a[high];
                a[high]=a[low];
                a[low]=temp;
            }
        }
        else {
            //其值居中者为a[(low+high)/2]
```

```
            temp= a[(low+high)/2];
            a[(low+high)/2]=a[low];
            a[low]=temp;
        }
    }
    for(int j=low+1;j<=high;j++) {
        if(a[j]<a[low]) {
            i++;
            if(i!=j) {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
    }
    temp=a[i];
    a[i]=a[low];
    a[low]=temp;
    return i;
}
void QuickSort(int a[],int low,int high) {
    if(low<high) {
        int w=Split(a,low,high);
        QuickSort(a,low,w-1);
        QuickSort(a,w+1,high);
    }
}
```

6.39

输入数组 `A[1...N]` 由N个等同的元素组成时，相当于 正序/倒序输入一个数组
这时候在执行SPLIT算法后，执行QUICKSORT递归算法相当于调用一颗 斜树

当划分的两个子数组里面存在0或者n-1个元素时，最坏划分情况就发生了，
最坏情形下，为正序或逆序排列，二叉树画出来应该是一棵斜树，并且需要经过n-1次递归调用才能完成，且第i次划分需要经过n-i次关键字的比较才能找到第i个记录，也就是枢轴的位置，所以：此时算法运行时间的递归式是T(n) = T(n-1) + T(0) + O(n) = T(n - 1) + O(n)此时解为O(n^2)。

因此，如果在算法的每一层递归上，划分都是最大程度的不平衡，那么，算法的时间复杂度是O(n^2)

比较次数，$1+2+3.....+n-1=n(n-1)/2$

即每次执行递归算法时每一个子序列都重新进行SPLIT算法，每个子序列都交换元素